(12) UK Patent Application (19) GB (11) 2 236 202 (13) A

(43) Date of A publication 27.03.1991

(21) Application No 9008244.7

(22) Date of filing 11.04.1990

(30) Priority data
(31) 411036 (32) 21.09.1989 (33) US

(51) INT CL[5]
G06F 11/30

(52) UK CL (Edition K)
G4A AFMD

(56) Documents cited
GB 2210482 A US 4849879 A

(58) Field of search
UK CL (Edition K) G4A AFMD AFMG AFMT
INT CL[5] G06F

(71) Applicant
Sun Microsystems Inc

(Incorporated in the USA - Delaware)

2550 Garcia Avenue, Mountain View, California 94043, United States of America

(72) Inventors
Martin Stanley Itzkowitz
Mark Liu

(74) Agent and/or Address for Service
Potts Kerr and Co
15 Hamilton Square, Birkenhead, Merseyside, L41 6BR, United Kingdom

(54) Extracting process performance information from a computer over a network

(57) A method and apparatus described allow the extraction of process specific performance information. An operating system module is supplied having procedures that service remote procedure calls. These procedures, when invoked, retrieve specific information from tables and registers in the operating system indicative of the status of resources which are used to manage the computer system. In a preferred embodiment, the remote procedure calls are executed by a monitoring process 80 on a second computer system 30 connected through a network 40 to a first computer system 10 on which the monitored process is executed, whereby the process and resource overhead on the first computer required to monitor process performance is minimized.

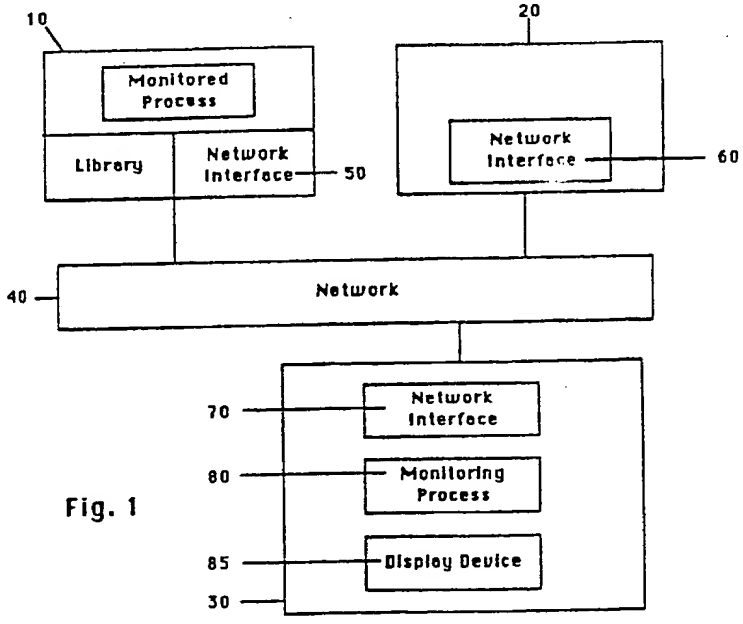

Fig. 1

At least one drawing originally filed was informal and the print reproduced here is taken from a later filed formal copy

GB 2 236 202 A

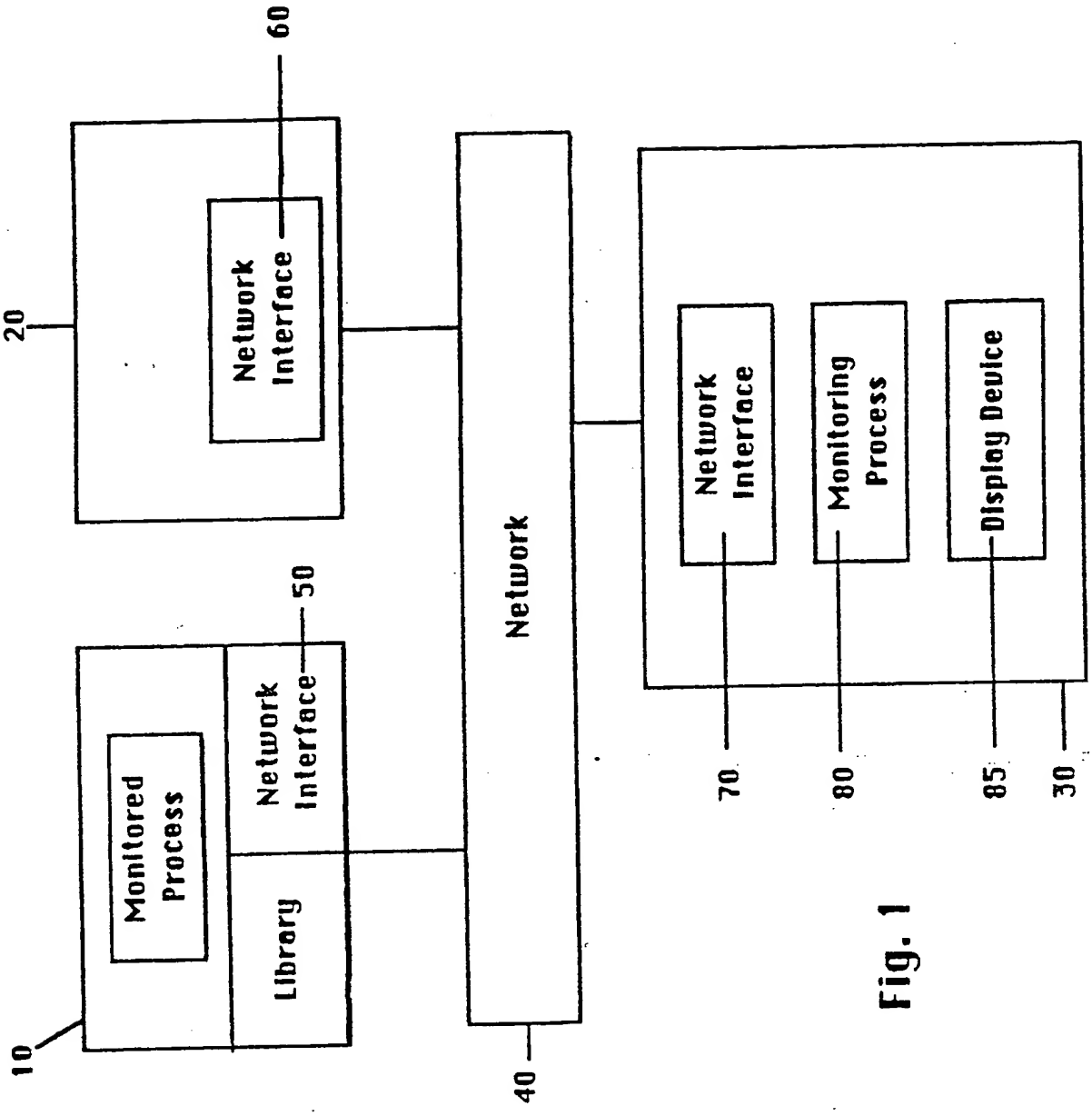
10
Monitored Process
Library
Network Interface
50
20
Network Interface
60
Network
40
Network Interface
Monitoring Process
Display Device
70
80
85
30

Fig. 1

```
#ifndef     lint
static      char sccsid[] = "@(#)sample_fetch.c 1.3 89/04/14 Copyr Sun Micro";
#endif

/*
 *          Copyright (c) 1989 by Sun Microsystems, Inc.
 */

#include            "wsmon.h"
#include            "st.h"
#include            <ski/ski.h>

/*
 *  sample_fetch.c
 *          routines for fetching rusage data from a sample
 */

/*
 *  sample_fetch_null
 *          fetch for unused variable — should never be called
 */

double                                                          sample_fetch_null
sample_fetch_null(samp_ent)
            Sample_entry_t          *samp_ent;
{
            terminate( FATAL, "sample_fetch_null invoked");
            return(0.0);

double                                                          sample_fetch_total_hard_faults
sample_fetch_total_hard_faults(samp_ent)
            Sample_entry_t          *samp_ent;
{
            double      value;

            value = (double) (samp_ent->usage_after.sru_majflt -
                        samp_ent->usage_before.sru_majflt);

            return(value);
}


double                                                          sample_fetch_total_soft_faults
sample_fetch_total_soft_faults(samp_ent)
            Sample_entry_t          *samp_ent;
{
            double      value;

            value = (double) (samp_ent->usage_after.sru_minflt -
                        samp_ent->usage_before.sru_minflt);

            return(value);
double                                                          sample_fetch_total_faults
sample_fetch_total_faults(samp_ent)
            Sample_entry_t          *samp_ent;
{
            double      value;

            value = (double) (samp_ent->usage_after.sru_majflt -

                        samp_ent->usage_before.sru_majflt +
                        samp_ent->usage_after.sru_minflt -
                        samp_ent->usage_before.sru_minflt);

           return(value);
double                                                          sample_fetch_rate_hard_faults
sample_fetch_rate_hard_faults(samp_ent)
            Sample_entry_t          *samp_ent;
{
            double      value;
            double      timediff;
            double      rate;

            value = (double) (samp_ent->usage_after.sru_majflt -
                        samp_ent->usage_before.sru_majflt);

            timediff = (double) (samp_ent->usage_after.sru_rtime.tv_sec -
                        samp_ent->usage_before.sru_rtime.tv_sec)+
                        (double) (samp_ent->usage_after.sru_rtime.tv_usec -
                        samp_ent->usage_before.sru_rtime.tv_usec)/1000000. ;

            if(timediff == 0.0){
                        rate = 0.0
            } else {
                        rate = value/timediff;
            }

            return(rate);
```

Fig. 2

1 of 7

sample_fetch_rate_soft_faults

```
double
sample_fetch_rate_soft_faults(samp_ent)
        Sample_entry_t      *samp_ent;
{
        double      value;
        double      timediff;
        double      rate;

        value = (double) (samp_ent->usage_after.sru_minflt -
                samp_ent->usage_before.sru_minflt);

        timediff = (double) (samp_ent->usage_after.sru_rtime.tv_sec -
                samp_ent->usage_before.sru_rtime.tv_sec)+
                (double) (samp_ent->usage_after.sru_rtime.tv_usec -
                samp_ent->usage_before.sru_rtime.tv_usec)/1000000. ;

        if(timediff == 0.0){
                rate = 0.0;
        } else {
                rate = value/timediff;
        }

        return(rate);
```

sample_fetch_rate_total_faults

```
double
sample_fetch_rate_total_faults(samp_ent)
        Sample_entry_t      *samp_ent;
{
        double      value;
        double      timediff;
        double      rate;

        value = (double) (samp_ent->usage_after.sru_majflt -
                samp_ent->usage_before.sru_majflt +
                samp_ent->usage_after.sru_minflt -
                samp_ent->usage_before.sru_minflt);

        timediff = (double) (samp_ent->usage_after.sru_rtime.tv_sec -
                samp_ent->usage_before.sru_rtime.tv_sec)+
                (double) (samp_ent->usage_after.sru_rtime.tv_usec -
                samp_ent->usage_before.sru_rtime.tv_usec)/1000000. ;

        if(timediff == 0.0){
                rate = 0.0;
        } else {
                rate = value/timediff;
        }

        return(rate);
```

sample_fetch_addressable_count

```
double
sample_fetch_addressable_count(samp_ent)
        Sample_entry_t      *samp_ent;
{
        double      value;

        value = (double) (samp_ent->usage_after.sru_np);
        return(value);
```

sample_fetch_in_core_count

```
double
sample_fetch_in_core_count(samp_ent)
        Sample_entry_t      *samp_ent;
{
        double      value;

        value = (double) (samp_ent->usage_after.sru_np_resident);
        return(value);
```

sample_fetch_referenced_count

```
double
sample_fetch_referenced_count(samp_ent)
        Sample_entry_t      *samp_ent;
{
        double      value;

        value = (double) (samp_ent->usage_after.sru_np_ref);
        return(value);
}
```

sample_fetch_modified_count

```
double
sample_fetch_modified_count(samp_ent)
        Sample_entry_t      *samp_ent;
{
        double      value;

        value = (double) (samp_ent->usage_after.sru_np_mod);
        return(value);
```

Fig. 2

sample_fetch_total_cpu_time

```
double
sample_fetch_total_cpu_time(samp_ent)
        Sample_entry_t      *samp_ent;
{
        double      value;

        value = (double) (samp_ent->usage_after.sru_utime.tv_sec -
                samp_ent->usage_before.sru_utime.tv_sec)+
                (double) (samp_ent->usage_after.sru_utime.tv_usec -
                samp_ent->usage_before.sru_utime.tv_usec)/1000000. +
                (double) (samp_ent->usage_after.sru_stime.tv_sec -
                samp_ent->usage_before.sru_stime.tv_sec)+
                (double) (samp_ent->usage_after.sru_stime.tv_usec -
                samp_ent->usage_before.sru_stime.tv_usec)/1000000. ;

        return(value);
}
```

sample_fetch_total_user_cpu

```
double
sample_fetch_total_user_cpu(samp_ent)
        Sample_entry_t      *samp_ent;
{
        double      value;

        value = (double) (samp_ent->usage_after.sru_utime.tv_sec -
                samp_ent->usage_before.sru_utime.tv_sec)+
                (double) (samp_ent->usage_after.sru_utime.tv_usec -
                samp_ent->usage_before.sru_utime.tv_usec)/1000000. ;

        return(value);
}
```

sample_fetch_total_sys_cpu

```
double
sample_fetch_total_sys_cpu(samp_ent)
        Sample_entry_t      *samp_ent;
{
        double      value;

        value = (double) (samp_ent->usage_after.sru_stime.tv_sec -
                samp_ent->usage_before.sru_stime.tv_sec)+
                (double) (samp_ent->usage_after.sru_stime.tv_usec -
                samp_ent->usage_before.sru_stime.tv_usec)/1000000. ;

        return(value);
}
```

sample_fetch_total_real_time

```
double
sample_fetch_total_real_time(samp_ent)
        Sample_entry_t      *samp_ent;
{
        double      value;

        value = (double) (samp_ent->usage_after.sru_rtime.tv_sec -
                samp_ent->usage_before.sru_rtime.tv_sec)+
                (double) (samp_ent->usage_after.sru_rtime.tv_usec -
                samp_ent->usage_before.sru_rtime.tv_usec)/1000000. ;

        return(value);
}
```

sample_fetch_rate_cpu_time

```
double
sample_fetch_rate_cpu_time(samp_ent)
        Sample_entry_t      *samp_ent;
{
        double      value;
        double      timediff;
        double      rate;

        value = (double) (samp_ent->usage_after.sru_utime.tv_sec -
                samp_ent->usage_before.sru_utime.tv_sec)+
                (double) (samp_ent->usage_after.sru_utime.tv_usec -
                samp_ent->usage_before.sru_utime.tv_usec)/1000000. +
                (double) (samp_ent->usage_after.sru_stime.tv_sec -
                samp_ent->usage_before.sru_stime.tv_sec)+
                (double) (samp_ent->usage_after.sru_stime.tv_usec -
                samp_ent->usage_before.sru_stime.tv_usec)/1000000. ;

        timediff = (double) (samp_ent->usage_after.sru_rtime.tv_sec -
                samp_ent->usage_before.sru_rtime.tv_sec)+
                (double) (samp_ent->usage_after.sru_rtime.tv_usec -
                samp_ent->usage_before.sru_rtime.tv_usec)/1000000. ;

        if(timediff == 0.0){
                rate = 0.0;
        } else {
                rate = 100. * value /timediff;
        }

        return(rate);
}
```

Fig. 2

sample_fetch_rate_user_cpu

```
double
sample_fetch_rate_user_cpu(samp_ent)
        Sample_entry_t      *samp_ent;
{
        double      value;
        double      timediff;
        double      rate;

        value = (double) (samp_ent->usage_after.sru_utime.tv_sec -
                samp_ent->usage_before.sru_utime.tv_sec)+
                (double) (samp_ent->usage_after.sru_utime.tv_usec -
                samp_ent->usage_before.sru_utime.tv_usec) /1000000. ;

        timediff = (double) (samp_ent->usage_after.sru_rtime.tv_sec -
                samp_ent->usage_before.sru_rtime.tv_sec)+
                (double) (samp_ent->usage_after.sru_rtime.tv_usec -
                samp_ent->usage_before.sru_rtime.tv_usec) /1000000. ;

        if(timediff == 0.0){
                rate = 0.0;
        } else {
                rate = 100. * value /timediff;
        }

        return(rate);
```

sample_fetch_rate_sys_cpu

```
double
sample_fetch_rate_sys_cpu(samp_ent)
        Sample_entry_t      *samp_ent;
{
        double      value;
        double      timediff;
        double      rate;

        value = (double) (samp_ent->usage_after.sru_stime.tv_sec -
                samp_ent->usage_before.sru_stime.tv_sec)+
                (double) (samp_ent->usage_after.sru_stime.tv_usec -
                samp_ent->usage_before.sru_stime.tv_usec) /1000000. ;

        timediff = (double) (samp_ent->usage_after.sru_rtime.tv_sec -
                samp_ent->usage_before.sru_rtime.tv_sec)+
                (double) (samp_ent->usage_after.sru_rtime.tv_usec -
                samp_ent->usage_before.sru_rtime.tv_usec) /1000000. ;

        if(timediff == 0.0){
                rate = 0.0;
        } else {
                rate = 100. * value /timediff;
        }

        return(rate);
```

sample_fetch_total_io_read

```
double
sample_fetch_total_io_read(samp_ent)
        Sample_entry_t      *samp_ent;
{
        double      value;

        value = (double) (samp_ent->usage_after.sru_inblock -
                samp_ent->usage_before.sru_inblock);

        return(value);
```

sample_fetch_total_io_write

```
double
sample_fetch_total_io_write(samp_ent)
        Sample_entry_t      *samp_ent;
{
        double      value;

        value = (double) (samp_ent->usage_after.sru_oublock -
                samp_ent->usage_before.sru_oublock);

        return(value);
}
```

sample_fetch_total_msg_read

```
double
sample_fetch_total_msg_read(samp_ent)
        Sample_entry_t      *samp_ent;
{
        double      value;

        value = (double) (samp_ent->usage_after.sru_msgrcv -
                samp_ent->usage_before.sru_msgrcv);

        return(value);
```

Fig. 2

sample_fetch_total_msg_write

```
double
sample_fetch_total_msg_write(samp_ent)
        Sample_entry_t      *samp_ent;
{
        double    value;

        value = (double) (samp_ent->usage_after.sru_msgsnd -
                samp_ent->usage_before.sru_msgsnd);

        return(value);
```

sample_fetch_rate_io_read

```
double
sample_fetch_rate_io_read(samp_ent)
        Sample_entry_t      *samp_ent;
{
        double    value;
        double    timediff;
        double    rate;

        value = (double) (samp_ent->usage_after.sru_inblock -
                samp_ent->usage_before.sru_inblock);

        timediff = (double) (samp_ent->usage_after.sru_rtime.tv_sec -
                samp_ent->usage_before.sru_rtime.tv_sec)+
                (double) (samp_ent->usage_after.sru_rtime.tv_usec -
                samp_ent->usage_before.sru_rtime.tv_usec)/1000000. ;

        if(timediff == 0.0){
                rate = 0.0;
        } else {
                rate = value/timediff;
        }

        return(rate);
```

sample_fetch_rate_io_write

```
double
sample_fetch_rate_io_write(samp_ent)
        Sample_entry_t      *samp_ent;
{
        double    value;
        double    timediff;
        double    rate;

        value = (double) (samp_ent->usage_after.sru_oublock -
                samp_ent->usage_before.sru_oublock);

        timediff = (double) (samp_ent->usage_after.sru_rtime.tv_sec -
                samp_ent->usage_before.sru_rtime.tv_sec)+
                (double) (samp_ent->usage_after.sru_rtime.tv_usec -
                samp_ent->usage_before.sru_rtime.tv_usec)/1000000. ;

        if(timediff == 0.0){
                rate = 0.0;
        } else {
                rate = value/timediff;
        }

        return(rate);
```

sample_fetch_rate_msg_read

```
double
sample_fetch_rate_msg_read(samp_ent)
        Sample_entry_t      *samp_ent;
{
        double    value;
        double    timediff;
        double    rate;

        value = (double) (samp_ent->usage_after.sru_msgrcv -
                samp_ent->usage_before.sru_msgrcv);

        timediff = (double) (samp_ent->usage_after.sru_rtime.tv_sec -
                samp_ent->usage_before.sru_rtime.tv_sec)+
                (double) (samp_ent->usage_after.sru_rtime.tv_usec -
                samp_ent->usage_before.sru_rtime.tv_usec)/1000000. ;

        if(timediff == 0.0){
                rate = 0.0;
        } else {
                rate = value/timediff;
        }

        return(rate);
```

Fig. 2

sample_fetch_rate_msg_write

```
double
sample_fetch_rate_msg_write(samp_ent)
        Sample_entry_t      *samp_ent;
{
        double      value;
        double      timediff;
        double      rate;

        value = (double) (samp_ent->usage_after.sru_msgsnd -
                samp_ent->usage_before.sru_msgsnd);

        timediff = (double) (samp_ent->usage_after.sru_rtime.tv_sec -
                samp_ent->usage_before.sru_rtime.tv_sec)+
                (double) (samp_ent->usage_after.sru_rtime.tv_usec -
                samp_ent->usage_before.sru_rtime.tv_usec)/1000000. ;

        if(timediff == 0.0){
                rate = 0.0;
        } else {
                rate = value /timediff;
        }

        return(rate);
```

sample_fetch_total_signals

```
double
sample_fetch_total_signals(samp_ent)
        Sample_entry_t      *samp_ent;
{
        double      value;

        value = (double) (samp_ent->usage_after.sru_nsignals -
                samp_ent->usage_before.sru_nsignals);

        return(value);
```

sample_fetch_total_vol_ctx

```
double
sample_fetch_total_vol_ctx(samp_ent)
        Sample_entry_t      *samp_ent;
{
        double      value;

        value = (double) (samp_ent->usage_after.sru_nvcsw -
                samp_ent->usage_before.sru_nvcsw);

        return(value);
```

sample_fetch_total_inv_ctx

```
double
sample_fetch_total_inv_ctx(samp_ent)
        Sample_entry_t      *samp_ent;
{
        double      value;

        value = (double) (samp_ent->usage_after.sru_nivcsw -
                samp_ent->usage_before.sru_nivcsw);

        return(value);
```

sample_fetch_rate_signals

```
double
sample_fetch_rate_signals(samp_ent)
        Sample_entry_t      *samp_ent;
{
        double      value;
        double      timediff;
        double      rate;

        value = (double) (samp_ent->usage_after.sru_nsignals -
                samp_ent->usage_before.sru_nsignals);

        timediff = (double) (samp_ent->usage_after.sru_rtime.tv_sec -
                samp_ent->usage_before.sru_rtime.tv_sec)+
                (double) (samp_ent->usage_after.sru_rtime.tv_usec -
                samp_ent->usage_before.sru_rtime.tv_usec)/1000000. ;

        if(timediff == 0.0){
                rate = 0.0;
        } else {
                rate = value /timediff;
        }

        return(rate);
```

Fig. 2

sample_fetch_rate_vol_ctx

```
double
sample_fetch_rate_vol_ctx(samp_ent)
        Sample_entry_t          *samp_ent;
{
        double      value;
        double      timediff;
        double      rate;
        value = (double) (samp_ent->usage_after.sru_nivcsw -
                samp_ent->usage_before.sru_nivcsw);

        timediff = (double) (samp_ent->usage_after.sru_rtime.tv_sec -
                samp_ent->usage_before.sru_rtime.tv_sec)+
                (double) (samp_ent->usage_after.sru_rtime.tv_usec -
                samp_ent->usage_before.sru_rtime.tv_usec)/1000000. ;

        if(timediff == 0.0){
                rate = 0.0;
        } else {
                rate = value /timediff;
        }

        return(rate);
```

sample_fetch_rate_inv_ctx

```
double
sample_fetch_rate_inv_ctx(samp_ent)
        Sample_entry_t          *samp_ent;
{
        double      value;
        double      timediff;
        double      rate;

        value = (double) (samp_ent->usage_after.sru_nivcsw -
                samp_ent->usage_before.sru_nivcsw);

        timediff = (double) (samp_ent->usage_after.sru_rtime.tv_sec -
                samp_ent->usage_before.sru_rtime.tv_sec)+
                (double) (samp_ent->usage_after.sru_rtime.tv_usec -
                samp_ent->usage_before.sru_rtime.tv_usec)/1000000. ;

        if(timediff == 0.0){
                rate = 0.0;
        } else {
                rate = value /timediff;
        }

        return(rate);
```

Fig. 2

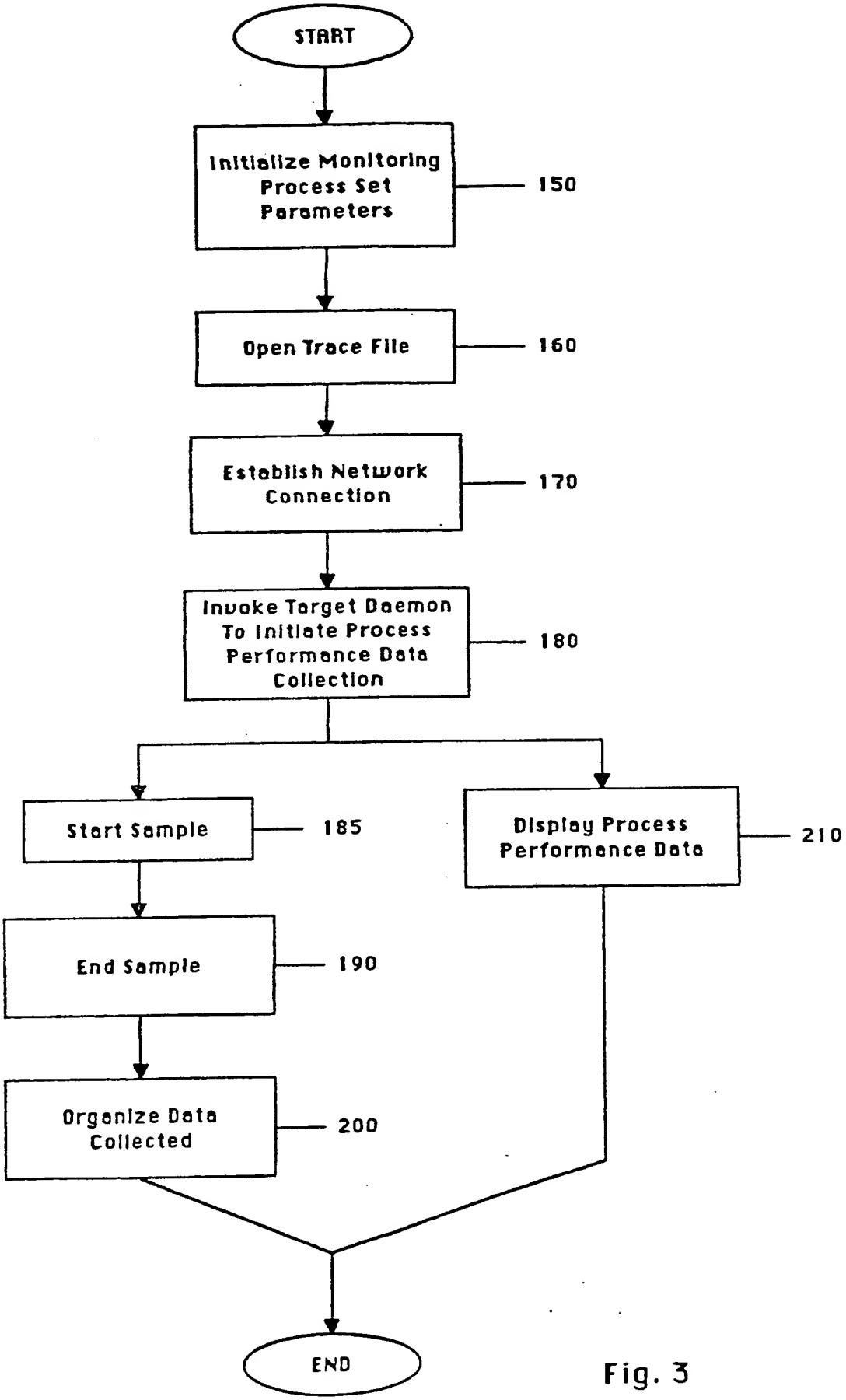
START
Initialize Monitoring Process Set Parameters
150
Open Trace File
160
Establish Network Connection
170
Invoke Target Daemon To Initiate Process Performance Data Collection
180
Start Sample
185
End Sample
190
Organize Data Collected
200
Display Process Performance Data
210
END

Fig. 3

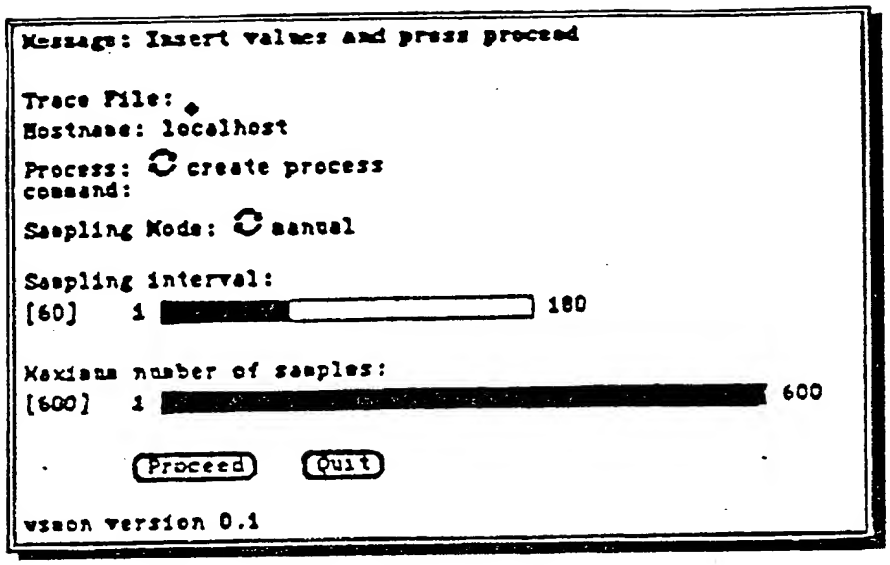
Message: Insert values and press proceed
Trace File:
Hostname: localhost
Process: create process
command:
Sampling Mode: manual
Sampling interval:
[60] 1 180
Maximum number of samples:
[600] 1 600
Proceed
Quit
vmon version 0.1

Fig. 4

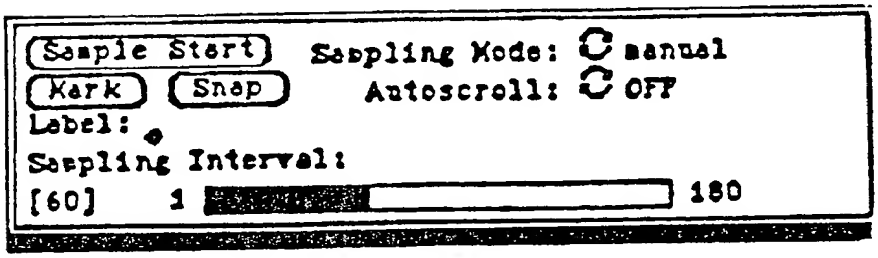
Sample Start
Sampling Mode: manual
Mark
Snap
Autoscroll: OFF
Label:
Sampling Interval:
[60] 1 180

Fig. 5

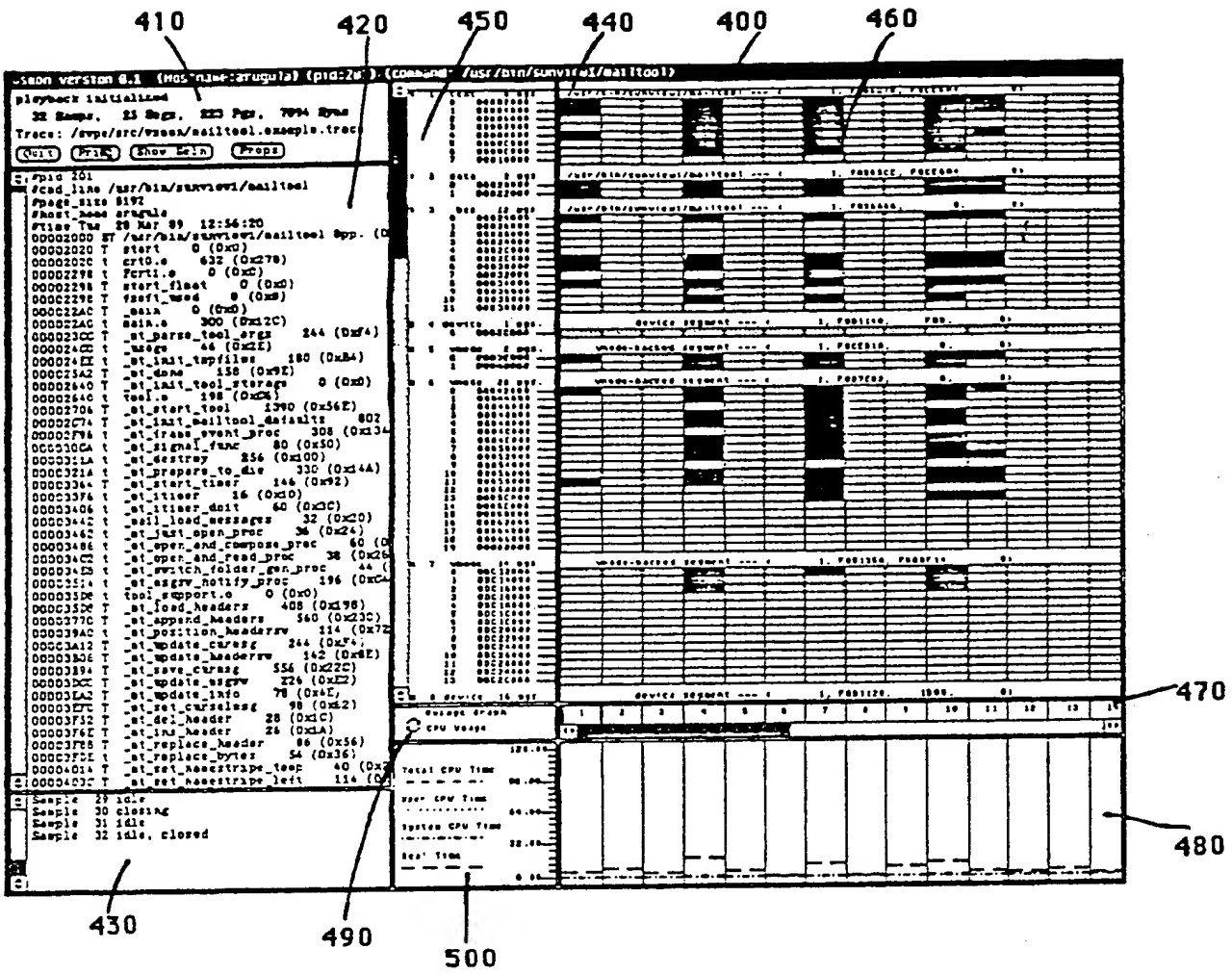
410
420
450
440
400
460
470
480
430
490
500

Fig. 6

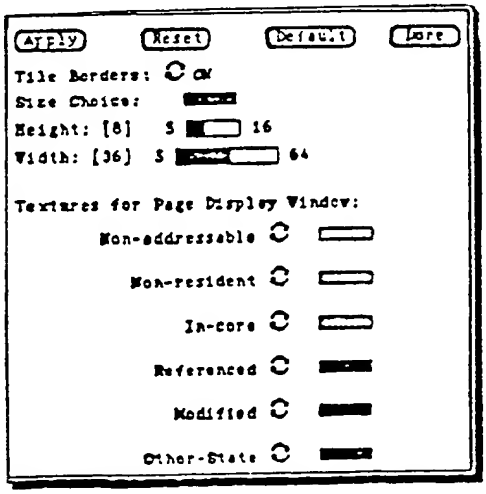
Apply
Reset
Default
Save
Tile Borders:
Size Choice:
Height: [8] 5 16
Width: [36] 5 64
Textures for Page Display Window:
Non-addressable
Non-resident
In-core
Referenced
Modified
Other-State

Fig. 7

METHOD AND APPARATUS FOR EXTRACTING PROCESS PERFORMANCE INFORMATION FROM A COMPUTER OVER A NETWORK

BACKGROUND OF THE INVENTION

1. FIELD OF THE INVENTION:

The method and apparatus of the present invention relates to the measurement of computer performance characteristics. More particularly, the method and apparatus of the present invention relates to the measurement of computer performance on a per-process basis.

2. ART BACKGROUND:

Frequently it is desirable to measure certain characteristics of a computer during normal operations in order to evaluate the performance of the computer or to provide a quantitative method of charging computer time. Typically, the amount of time or the percent of time the CPU is used during a sample period, the number of input/output functions executed or the number disk accesses which occurred are measured and evaluated to be indicative of system performance. To understand the performance of a particular application running on a computer system, more detailed information is required, such as the distribution of CPU time spent over the program's address space for a particular operation (as opposed to the entire run of the program), or information about which pages of the application's address space are referenced or modified to perform a particular operation.

In one method to measure system performance, a process is initiated on the computer to be monitored and certain characteristics of the computer are measured. However, the monitoring process itself contributes to the overhead of the computer and the process resource usage. Therefore any measurements recorded are distorted by the execution of the monitoring process.

In another method, after a process has completed execution, a subsequent process can be initiated to read certain information from the operating

system relevant to system performance. However, the characteristics measured relate to the performance of the computer as a whole for the entire time period of execution. The characteristics are not broken down and allocated to the processes being executed by the computer during the measurement time period. Thus, such measurements cannot be analyzed according to each process. Furthermore, the effect of a certain process on the performance of other concurrently executing processes can not be determined.

The above monitoring techniques are further limited by the processes that can access the information. Performance information is only accessible by the process itself being monitored or a process which created the monitored process executed. Thus a non-related process as well as a process executing on a remote device can not access the performance information.

One important system characteristic relevant to system performance in a virtual memory system is memory usage. A single process' memory usage may be identified by its working set and resident set. The term working set is used to describe the set of all pages referenced by a process during a time interval, or during an operation. The term resident set is used to describe the set of all pages belonging to the process that is resident in memory during a time interval or operation.

The resident set represents the operating system's response to the application's working set behavior. When a page that is not in the process' resident set is referenced, the operating system reads it in from the disk, a relatively expensive operation. When a page is no longer being referenced, it may be dropped from the resident set. If the page has not been modified (written), the physical memory it occupies is released for use to hold another page for the same or another process. If the page was modified, it must be written to disk before it can be released, also an expensive operation. For further information on paging, see Bach, The Design Of The UNIX Operating System (Prentice-Hall 1986).

Optimal program behavior implies minimal resource utilization. For the memory resource, this means that the program has as small a resident set as possible, and that each page in the process' address space has few transitions in or out of the resident set. Furthermore, optimal behavior minimizes the number of pages that are modified; data and variables that are likely to be changed during the running of the program should be grouped together so that the system need not write out several pages, each of which only has one modified piece of data.

When tuning an application program for optimal running on a computer, the programmer has no direct control over the resident set, because it is the result of interactions between the application's working set behavior and the operating system's paging strategy. The programmer does have control over the working set, and should tune the application to minimize the working set in the expectation that doing so will also decrease or minimize the resident set and therefore free up more memory for other applications.

Therefore in order to best optimize a process, a programmer should have the ability to review system performance information with respect to an individual process. In addition, the programmer should be able to review the performance information during specified sampling intervals during the execution of the monitored process as well as the performance characteristics at the completion of execution of the process.

SUMMARY OF THE INVENTION

It is therefore an object of the present invention to provide a method and apparatus for the measurement of computer performance characteristics on a per-process basis.

It is an object of the present invention to provide a method and apparatus for the measurement of computer performance characteristics without the distortion of performance characteristics caused by the concurrent execution of the process performing the measurements.

It is an object of the present invention to provide a method and apparatus for the measurement of process performance characteristics during specified sampling periods during the execution of the monitored process.

It is further an object of the present invention to provide a method and apparatus for the measurement of computer performance characteristics across a network, whereby the performance characteristics of a first computer connected to the network are measured by a second computer connected to the network.

It is an object of the present invention to provide a method and apparatus to read and record an individual process' working set.

Utilizing the method and apparatus of the present invention a process can monitor the performance characteristics of individual or multiple non-related processes on a per-process basis.

In one embodiment, the monitoring process and the monitored reside on separate computers connected through a network. Thus, a first computer connected to a network through a general network interface can monitor the performance

characteristics of a second computer connected to the network, similarly connected through a general network interface, on a process by process basis. The performance characteristics of the monitored or target computer, such as performance characteristics relative to memory operations, input/output operations, or the occurrence of certain events, for example, context switches, can be easily determined by the monitoring computer connected to the monitored computer through the network.

A library of operating system routines which service remote procedure calls is established on the target computer. These routines encompass basic functions such as "Return Resource Usage", "Return Address Space Data", "Start Virtual Page Monitoring", "Clear Virtual Page Reference/Modify Bits" and "Stop Virtual Page Monitoring". The routines can access data within the operating system and are invoked by remote procedure calls transmitted by a monitoring process which may be located on a remote computer connected through a network. Through the remote procedure calls, the monitoring process can retrieve raw performance data from the monitored process to subsequently analyze and organize the information to generate the performance characteristics of the monitored process.

In another embodiment, the monitoring process and the monitored process reside on the same computer whereby process performance characteristics of the monitored process can be measured during specified sampling intervals during the execution of the monitored process.

In another embodiment, the process to be monitored is modified to make the same remote procedure call that a separate monitoring process would make, and to record the data in the same form that a separate monitoring process would, thereby, in effect, monitoring itself.

BRIEF DESCRIPTION OF THE DRAWINGS

The objects, features and advantages of the method and apparatus of the present invention will be apparent from the following detailed description of the preferred embodiment in which:

FIG. 1 is a block diagram illustration of computers connected through a network in accordance with a preferred embodiment of the present invention.

FIG. 2 is a listing of exemplary code for the retrieval and interpretation of raw data.

FIG. 3 is a flowchart illustrating the preferred embodiment of the process employed in the method and apparatus of the present invention.

FIG. 4 illustrates a initialization panel employed in the preferred embodiment of the present invention.

FIG. 5 illustrates a collection control window employed in the preferred embodiment of the present invention.

FIG. 6 illustrates the display generated containing the process performance information according to the preferred embodiment of the present invention.

FIG. 7 illustrates the properties window of the display utilized in the preferred embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

The method and apparatus of the present invention is a performance based tool for observing the behavior of a process. The performance tool is designed to allow a software developer or user to understand the usage of memory and other resources by an application. It is specifically designed to monitor process information relative to individual processes, rather than system-wide, multiple process information. In one embodiment, the method and apparatus of the present invention provides the means for visualizing the program's working set as a function of time or on a per-operation basis. The information collected consists of snapshots of resource consumption and reference/modification information for each page in the process' address space of the virtual memory system.

A system module of procedures is created and added to the operating system on the machine containing the process to be monitored. The system module contains the procedures which execute or service Remote Procedure Calls. The procedures perform simple operations of reading specified tables or registers relevant to the monitored process and returning the information read. These procedures are called or initiated by the monitoring process through a network interface on the computer and the information retrieved is returned to monitoring computer for subsequent organization and analysis of the data. No special network or network interface is required; a general purpose network and network interface employed in a typical computer environment which support remote procedure calls may be used.

A typical computer network is illustrated in the block diagram of Fig. 1. Fig. 1 shows a plurality of computers 10, 20, 30 connected via a network 40. Each computer contains a mixture of hardware and software which functions as the network interface 50, 60, 70 which formats data/information for transmission on the network as well as receives such formatted information from another computer connected to the

network and removes the network specific information leaving the original data/information for use by the computer.

In the present example, computer 10 is the computer system containing the process or processes to be monitored. Computer 30 is the computer containing the process which monitors process specific performance information on computer 10. Operating on computer 30 is the monitoring process 80 which sends out remote procedure calls to the computer 10 requesting specific performance data regarding the process, and receiving the data transmitted back from the computer 10 containing the monitored process. The raw data received is then analyzed and organized for presentation to the user. Preferably the information is presented to the user in graphical form over display 85 for easy visual comprehension by the user.

The user controls, through the monitoring process, the process performance data to be collected and the time periods during which the data is collected. Thus the user can access performance data reflecting time periods during the execution of the monitored process and not just cumulative performance data accessible only after the monitored process has completed execution. In addition, the user can specify how the data is to be organized, analyzed and presented to the user for review. By providing these features and functionality in the monitoring process instead in the monitored process, a majority of the system overhead required to monitor performance information in the monitored process is eliminated. Thus the performance analysis is more accurate in as much as system overhead not normally present (when not monitoring system performance) in the operating environment is minimized.

Although the computer system requires a network interface to receive and initiate the servicing of remote procedure calls, it is not necessary to physically connect the computer system to the network. The remote procedure calls may be issued by a process executing on the same computer system as the monitored

process. The network interface will interpret the recipient system of the remote procedure call to be one in the same computer system and will direct the call to the system module located in the operating system to service the call.

Thus, the system module containing the procedures which service remote procedure calls may also be utilized by a separate, unrelated monitoring process executing on the same computer system. Similarly, the remote procedure calls may be executed by the monitored process. Sometimes it is preferable to issue the remote procedure calls from the monitored process when there is a requirement to precisely synchronize the extraction of performance data with the specific sections or lines of code in the monitored process.

The embodiment described is described with reference to computer systems operating in a UNIX® operating system environment (UNIX is a registered trademark of AT&T). However, it is well understood by those skilled in the art that the present invention is not limited to a specific operating system and can be applied to any networked computer systems.

The monitoring process controls the data acquisition on the computer system containing the monitored process through commands or procedure calls to execute certain procedures located in a system module in the operating system of the computer. These commands, referred to as "Remote Procedure Calls", enable the monitoring process to simply and easily invoke procedures in the system module. Typically, remote procedure calls are used to enable a process executing on a first computer system to invoke procedures or process in a second remote, but connected computer system. The procedures to service the remote procedure calls (i.e., the processes executed in response to remote procedure calls) are located in a system module which is added to the operating system, or in the case of the UNIX operating system, to the kernel. The remote procedure call system is enabled by a system background process (referred to as a "daemon process" in UNIX); once enabled, the

requests are processed directly by the Kernel procedures in the system module. For information on remote procedure calls, see "Remote Control", BYTE, p. 235-240, July 1989; Sun Microsystems, Inc, "Remote Procedure Call Programming Guide", Revision A, May, 1988.

Alternatively, as will be discussed below, system calls may be issued by a process to start and stop virtual page monitoring of the same process or another process on the same computer system. The operating system includes a system module which responds to and services the system calls and which contains procedures to service system calls related to virtual page monitoring.

Preferably, the remote procedure call is in the form: CALLRPC (parameters), where the parameters include the procedure name or identification and the identification of the remote computer on which the procedure is to be executed. The network interface, upon receiving a remote procedure call from a process, will format the call in a predetermined message format and transmit the message to the computer system connected to the network which is specified in the remote procedure call. To execute a procedure on the same computer from which the remote procedure call is issued, the parameter identifying the remote computer is input to be the same computer issuing the call. The network interface analyzing the call, to determine where to transmit the call to, will determine that the call is to be serviced on the same computer. Thus, the call will not be transmitted out over the network, but will be transferred to the system module containing the procedure to service the call.

Upon receipt of the remote procedure call, the procedure servicing the remote procedure call begins execution and transmits messages to the process which issued the call. The messages are preferably in a format such as the XDR format utilized by Sun Microsystems, Inc. (See Sun Microsystems, Inc., "Remote Procedure Call Programming Guide", pp. 97-142, Revision A, May, 1988.). At least one message

is transmitted signaling that execution of the procedure is complete. Additional messages may be transmitted providing performance data to the monitoring process.

In the preferred embodiment, procedures are provided to start and stop sampling periods during the execution of the monitored process in order to retrieve non-cumulative process performance data during different portions of the process.

The procedures located in the system module preferably include "Return Resource Usage", "Start Virtual Page Monitoring", "Return Address Space Data", "Clear Virtual Page Reference/Modify Bits" and "Stop Virtual Page Monitoring".

The procedure "Return Resource Usage" returns the accumulated resource usage read from the portion of the operating system's memory containing information for the process. Preferably, in a UNIX based system, the raw data returned is a superset of the information maintained as a "rusage" structure in the kernel with a time stamp indicating the time the data was retrieved (i.e., when the remote procedure call was processed). The resource data returned is that data stored in tables in the operating system's memory which contain accumulated accounting information for the operating system. The data maintained is dependent upon the operating system. For example, in the UNIX operating system, the system data can be retrieved using the system call "GETRUSAGE" and the following information is retrieved:

Real time (Accumulated real time)
User Time (Accumulated time of user process execution in CPU)
System time (Accumulated time of system process execution in CPU)
Number of Page Faults
Number of Swaps
Number of blocks read from I/O device
Number of blocks written to I/O device
Number of accumulated signals

Number of voluntary context switches

Number of involuntary context switches

Number of messages sent

Number of messages received

This procedure is preferably invoked at both the beginning and end of a sample period. The difference between the values at the start of the sample period and at the end of the sample period are the resources used during the sample period. Exemplary code for the retrieval and interpretation of the resource usage data is set forth in Figure 2.

The procedure "Start Virtual Page Monitoring", creates a duplicate or back copy of a portion of the page table of the monitored process (in UNIX the table is located in the kernel). Each page table entry contains the physical address of the page, protection bits indicating whether processes can read, write or execute from the page, and the following bit fields: "valid", "reference", "modify", "copy on write" and "age". The operating system (in UNIX, the kernel) turns on the valid bit to indicate that the contents of a page are legal. The reference bit indicates whether a process recently referenced a page, and the modify bit indicates whether a process recently modified the contents of a page. The copy on write bit, used in the fork system call, indicates that the kernel must create a new copy of the page when a process modifies its contents. The age bits indicate how long a page has been a member of the working set of the process. Whenever the reference or modify bits in the page table are changed according to memory read, write and paging operations, the changes are subsequently recorded in the back copy. The back copy is accessible and controllable by the monitoring process throughout the procedure. Thus, the back copy can be cleared and read when required by the procedure without affecting the page table. Furthermore, the initialization or the clearing of the page table by the system does not affect the back copy thereby allowing the independent collection of cumulative data.

The procedure "Return Address Space Data" returns the data contained in the back copy of the paging tables as well as information from the original page table regarding the resident status of the page. For each page in the process' address space, bits are returned indicating whether or not the page is resident in memory at the time of the request, and whether the page has been referenced or modified since either the creation of the back copy or the clearing of the back copy (by the command "Clear Virtual Page Reference/Modify Bits " described below). In addition, information is retrieved from the original page table indicating whether the page is resident in memory.

The procedure "Clear Virtual Page Reference/Modify Bits" is used to clear the reference/modify bits in the back copy of the page table. The procedure is invoked at the beginning of a sample period to initialize the table.

The procedure "Stop Virtual Page Monitoring" is invoked at the end of the monitoring process to stop virtual page monitoring and release the operating system memory used to create the back copy of the page table.

As discussed earlier, the procedure calls related to virtual page monitoring, that is, "Start Virtual Page Monitoring", "Return Address Space Date", "Clear Virtual Page Reference/Modify Bits" and "Stop Virtual Page Monitoring" may alternatively be implemented as system calls. The operating system module(s) containing system calls is supplemented to include procedures which are responsive to and service the system calls related to virtual page monitoring. These system calls and corresponding procedures may be invoked by the monitored process or a monitoring process executing on the same computer system as the monitored process.

The monitoring process will now be described with reference to Fig. 3. At block 150 the monitoring process is initialized and the monitoring parameters are set. Preferably an initialization panel, such as the one illustrated in Fig. 4 is used. The user

initiating the monitoring process inputs the parameters for the monitoring process. Preferably the initialization parameters include, but are not limited to:

Trace file - a string giving the name of the file for the data collected ,

Host - a string giving the name of the host computer on which the process to be monitored resides.

Process - a cycle item used to specify the process to be monitored. It cycles between "create process" (if the monitored process is to be created) or "attach to pid [process id]" (if the process has already been created).

Command - A string giving the name of the command executed by the process being monitored.

Sampling mode - a cycle item for selection of the operating mode for sampling. It cycles through "manual" (for user-directed sampling), "interval" (for taking periodic samples of data) or "playback" (for displaying a previous sample).

Maximum number of samples - the maximum number of samples to be read.

Proceed - a button used to indicate that the user has completed entry of the parameters, and that the sampling can start.

Quit - used to exit the program.

After the input parameters have been entered, at block 160, a file is open to receive the monitoring information ("the trace file") and, at block 170, the network connection between the monitored process and the monitoring process is established. In the present embodiment, a daemon (a daemon is a system background process) initiates a process on the computer system containing the process to be monitored, which attaches to or creates the process to be monitored and enables access to the descriptive header and symbolic reference information of the process (the loadmap) which is subsequently used in organizing the process specific data. In the UNIX operating system environment, the system call "PTRACE" enables a first process to attach to a second process whereby the first process attains access to the header and symbolic information of the second process. The process transfers this loadmap information back to the monitoring process which writes it back into the trace file and

maintains a copy to correlate the symbolic references with address space data subsequently received.

At block 180, a connection to the monitoring daemon on computer 10 is made and remote procedure calls are issued to begin collection of data. First a call of the type "Start Virtual Page Monitoring" is made to initialize the system. Then the monitoring process continues with two sets of operations that can proceed in parallel. The first set consists of blocks 185 and 190 for collecting sample data, and block 200 for organizing the sample data. The second set consists solely of block 210 for displaying the performance data.

At block 185, a sample is started by using the two remote procedure calls "Clear Virtual Page Reference/Modify Bits" and "Return Resource Usage". At block 190, a sample is terminated by using the two remote procedure calls "Return Address Space Data" and Return Resource Usage". At block 200, data from any newly-collected samples is processed to convert it into a format more suitable for displaying.

The address space information is scanned to calculate cumulative counts of pages addressable, resident, referenced and modified, and to see if any new regions of address space have been created by the program. If any are found, their parameters (size, type) are added to the Namelist display described below (window 420, Fig. 6). Each of the usage parameters returned by the remote procedure call "Return Resource Usage" is examined to determine a maximum value for that parameter over all samples. Furthermore, since a time stamp indicating the time at which data was collected is attached to each sample, a rate of usage can be calculated for each of the cumulative usage parameters . From this data, any of the resource usage displays, illustrated in Fig. 6 and discussed below, can be shown (see window 480, Fig. 6).

In a preferred embodiment, process performance data can be collected during predetermined portions of the execution of the monitored process. For example, performance data can be collected during the execution of certain routines or certain input/output processes. The samples to be collected are specified by the monitoring process. For example, a sampling rate may be preprogrammed into the monitoring process (i.e., default values), or determined by the user when the monitoring process is initiated. Alternatively, the samples may be selected during the execution of the monitoring process.

Preferably the sampling is controlled by means of a collector control window, such as the one illustrated in Fig. 5, which contains various options used for controlling the collection of data. This window is located on the display screen during the execution of the monitoring process for easy access by the user. The window may contain such items as:

Sample - This is a button used to manually start and stop a sample interval. The button will show either "Sample Start" or "Sample End" depending upon whether the monitoring process is between samples or in the middle of a sample.

Snap - A button used to end one sample and start the next sample with no gap between samples.

Mark - A button used to add a flag to the current sample label to mark a point of interest to the user.

Autoscroll - A cycle to enable and disable automatic scrolling of the display of calculated data.

Sampling Modes - A cycle item used to switch between a manual mode, in which the user indicates manually using the "sample" button the start and end of a sample, and an interval mode in which samples are taken sequentially for an interval of duration specified by the "Interval" function.

Interval - A slider function used to set the size of a sampling interval in seconds. Its current value is used at the start of each interval when in the interval sampling mode.

Sample Label - A text string used for labeling the current sample when in the manual sampling mode. At the end of each sample, the current contents of the sample label string are used as the label for the sample and the contents are cleared to allow the user to enter a label for the next sample.

The program sampling mode, as determined at initialization or through the control window, can be either manual or interval. In manual mode, samples are taken in response to user selection or "clicks" on the "sample" button in the collector control window. A sample is defined by two clicks. The first click marks the start of the interval and the second click marks the end of the interval. Samples can be of any time duration, and there is a label associated with each interval (as stored in the "sample label field" in the collector control window) for subsequent reference.

In the interval mode, the monitoring process records data at fixed time intervals, and generates a label for each sample, preferably indicative of the location of the sample in the sampling sequence. The sampling interval can be as short as one second, or as long as 180 seconds.

As the samples are collected, the data is stored in a trace file. The trace file consists of a series of blocks, encoded in XDR format, which stores the data passed between the kernel and the monitoring process. The first block in a trace file is the loadmap from the monitored process. Subsequent blocks contain sample information, with each sample represented by five blocks, a sample header block, a sample label block, a block containing the data returned from the "Return Resource Usage" call at the start of the sample, a block containing the data returned by the "Return Resource Usage" call at the end of the sample, and a block containing the data returned by a "Return Address Space Data" call at the end of the sample.

Preferably the process performance data is "massaged" or converted into data easily reviewable by the programmer/reviewer. It is also preferred that the data is converted into graphical data for output on a display device such as a high resolution graphics device. An illustrative example of a generated display for a single process is shown in Fig. 6. Referring to Fig.6, the name stripe 400 identifies the window as running the monitoring process and contains information about the process being monitored such as the machine on which the monitored process is running (hostname), the process id of the monitored process (pid) and the command line for the monitored process which indicates the computer program being executed by the monitored process (command). The control and status panel 410 presents three lines of status information, The first line gives the current status of operation. The second line contains a summary count of the number of samples taken, the logical address segments, pages and symbols utilized by the monitoring process. The third line specifies the name of the trace file. The fourth line contains four buttons which are used to specify operations. The "Quit" button causes the termination of the monitoring process. The "Print" button causes the generation of a hard copy image of the current display. The "Show Sel" button forces a selected page, symbol, and sample to be visible in all of the windows so that all performance data relevant to a selected symbol or a selected page during a particular sample taken can be reviewed on the display which assists the programmer in correlating the performance data collected to the source code of the monitored process. Selections are made by moving the cursor to the line containing the symbol of interest in the Namelist window 420, or the line showing the page of interest in the address space display (windows 440, 450 and 460). The "Props" button causes the posting of a property sheet or window used to change the appearance of the address space display as is described below with reference to Fig. 6. For example, the vertical size of the tiles used in the address space display could be made smaller to fit more of the address space on the display, or larger so that the user can see differences between tiles more easily. Similarly, the color or texture used to represent the state of a page could be changed so as to not

distinguish between referenced and modified pages, for those cases where the distinction is not of interest to the user.

The name list window 420 is a scrollable text sub-window which contains a table of the form:

Virtual Address	Type	Symbol Name	Size In Bytes

The Name List Window is generated from symbolic references acquired by the loadmap at the start of the measurement process and from the address segment information retrieved by each "Return Address Space Data" call made during sampling which indicates whether any new symbolic references had been created. There is one row for each symbol in the program's address space including all statically and dynamically loaded symbols but preferably excluding de-bugger symbols generated by a debugging program typically employed by programmers to debug or to eliminate errors in software. The symbols are identified from the loadmap copied into the trace file. Special entries are inserted at every logical segment boundary which identify the segment as text, data, BSS, V-node or device segments. At the top of the name list are several lines each beginning with a # sign giving information about the process being monitored including its process ID, the command name used to initiate process, the host name of the computer containing the monitored process, the time the monitored process was initiated, and the page size. Additional lines are added to provide information read from the header information of the executable image of the monitored process. Specifically it is noted whether if the program has been statically linked, if it has been stripped of symbols or if there was an error generating the map. Using the name list window, further information may be highlighted relative to a particular symbol on the display. A click anywhere on a row in the window will select a symbol, resulting in the row corresponding to the symbol selected to be shown in reverse video in the name list window and the page containing the symbol to be highlighted in the address space portion of the display 440.

The sample label window 430, is another text sub-window that shows the labels of various samples. Each label is obtained from the sample data, which in turn has been entered by the user at the time the sample is terminated. The label is inserted without change, and is used to identify the particular sample to the user. A click on the row in the sample label window will select that sample and it will cause the address space and resource usage displays to scroll horizontally so that information relative to the selected sample is visible in the respective displays. The line containing the selected sample is then underlined in the sample label window. Thus the programmer can easily see the sample and the state of the process' pages during that sample, and the various resources consumed during that sample.

The address space display 440 comprises side-by-side windows, page address window 450 and page display window 460. Each row of the page address window 450 corresponds to a single memory page and displays the virtual address, in hexadecimal, of the page and the page offset relative to the logical segment containing the page which was retrieved through the remote procedure call "Return Address Space Data". A tick mark is placed next to the center of the corresponding row of tiles in the page display window 460 for easy cross-reference. Lines corresponding to segment boundaries provide the relative index of the segment the type (Text, Data, BSS, V-node or device) and the number of pages in the segment. The page address window 450 has a vertical scroll bar which controls the scrolling of the display window and allows the user to view different regions of the address space.

The page display window 460, which is positioned to the right of the page address window 450, consists of an array of tiles, scrollable in horizontal and vertical directions. The raw data used to determine the state of a particular page in a sample is obtained as part of the data returned by the "Return Address Space Data" remote procedure call. The data is processed to give a state code for each page, which is then mapped into a color or texture pattern for the tile corresponding to that page. Each tile is preferably grey textured for a monochrome display or colored for a

color display giving the state of one page in the process' address space during one sample interval. The correspondence between the texture or color of the rectangle and the state of the particular page is given by a legend on the page display window property sheet. Each row of tiles displays the states of a particular page during the various samples taken in the temporal order of the samples wherein one tile represents the state of a page during one sample. The length of time represented by each sample is identified on the CPU usage graph described below. The page display window is horizontally scrolled according to the time period selected in the time access window. Each column of tiles provides the state of various pages in the process' address space during one particular sample. The vertical axis for the page display window 460 is tied to the page address window 450 and its scroll bar to provide easy manipulation and indexing of data. In as much as pages are grouped into segments, the tiles are grouped into blocks according to the corresponding logical segments in the process' address space.

A line of text is placed above each block of pages identifying the name of the file corresponding to the logical segment and a set of 4 32-bit numbers which is the kernel identification numbers for the logical segment which is part of the information returned by the remote procedure call "Return Address Space Data". Clicking or selecting on any line in the page address or page display window will select a page and cause the data relevant to the selected page to be highlighted in the page address window 450, the display window 460 and the name list window 420. In addition, the name list window 420 will be scrolled to display, highlighted, the first symbol located in the page.

As described above, the state of each page is shown during the samples by the color or texture of the corresponding tile in the page display window 460. The page state information is determined by examining the information retrieved by the "Return Address Space Data" remote procedure call which indicates whether a page

has been referenced or written (modified), resident in memory or addressable. A page may be in one of the following states:

Non-addressable - the page cannot be referenced by the program. Normally, non-addressable pages are omitted from the display but are shown if a page is not addressable during some samples but addressable during others. This may occur, for example, when a process' heap increases in size during one particular sample. The newly added pages will be shown as non-addressable during samples taken prior to the increase in the heap size.

Non-resident - the page is addressable but not resident in memory.

In-core - the page was resident in memory but neither referenced nor modified by the application during the sample.

Referenced - the page was referenced but not modified by the application during the sample.

Modified - the page was modified by the application during the sample.

Other - the page was in an unknown state.

The size, textures or colors used for tiles in the page display window 460 may be set by using the page display window property sheet illustrated in Fig. 7. The page display window property sheet is invoked by a button selection on the "props" button in the control and status window 410. The property sheet, illustrated in Fig. 7, has the following items in it:

Apply - a button to cause the transfer of parameters as shown on the property sheet to the main display.

Reset - a button to cause the reset of the property sheet values to the values currently used in the main display.

Default - a button to reset the property sheet values to the default values.

Done - a button to cause the closing of the property sheet window.

Tile borders - A toggle switch which turns on/off the display of borders around each tile. The tiles may be shown in varying color or texture optionally with a solid line border drawn around each tile.

Size choice - a sample tile that is modified in size, using the height and width sliders, showing a tile of the size on the display.

Height - a slider type control used to set the height of a tile.

Width - a slider type control used to set the width of a tile.

Buttons are also provided to select the color and/or texture to correspond to a particular page state.

The time axis window 470 identifies and provides a legend for the horizontal axis of the page display window 460 and the resource usage window 480. It is divided into equal size sections corresponding to individual samples taken, which may or may not represent equal size time intervals. Each section is labeled with a sample number which are sequentially assigned in the order generated. The time axis window 470 has a scroll bar in the horizontal direction that scrolls itself as well as the page display window 460 and resource usage window 480. A click on a window indicative of a sample within the time access window 470 selects that sample. The sample label window 430 will also be scrolled, if necessary, so that the selected sample is visible in the sample label window. The selected window (sample) is highlighted by a bolder border in the time access window 470 and its label is underlined in the sample label window 430.

The resource usage display consists of three sub-windows below and to the left of the time axis window 470. Immediately to the left of the time access window 470 is a panel 490 containing a button used to specify which of the nine available resource usage graphs will be displayed. Immediately below the button is a scale window 500 which contains the scale and legend for the chosen graph. To the right of the legend and scale is the resource usage window 480 containing graphical plots of resource usage (either per second averages or totals for each sample). Each graph contains several plots shown as lines of different textures or colors sharing the same vertical scale.

The nine resource usage graphs that can be displayed are: the paging graph, the paging rate graph, the page residency graph, the CPU usage graph, the CPU rate graph, the I/O usage graph, the I/O rate graph and the miscellaneous graph, the miscellaneous rate graph. The raw data used to generate the graphs are stored in the trace file at the time the data is retrieved. Exemplary code for the generation of the data is set forth in Figure 2.

The paging graph shows the number of total faults within a sample interval, total hard faults and total soft faults. Hard faults are those which require physical I/O to get the page in, soft faults do not require physical I/O but only a remapping of the page from the free list or restoring the mappings in the MMU. The paging rate graph shows hard faults per second within the sample interval, soft faults per second and total faults per second. The values (counts) are obtained by subtracting the respective cumulative values from the data returned by the "Return Resource Usage" remote procedure call at the start and end of the sample. The rates are computed by dividing the counts by the real time duration of the sample which is computed by subtracting the time stamps of the two "Return Resource Usage" calls.

The page residency graph displays four rows of information: total addressable pages, total in-core pages, total referenced pages and total modified pages. In the page residency graph the in-core count includes referenced and modified pages wherein referenced includes modified (by contrast in the page display window in-core excludes referenced or modified pages and referenced excludes modified). The data is obtained by scanning the data returned by the "Return Address Space Data" remote procedure call at the end of each sample, and counting the pages in each state.

The CPU usage graph shows four rows of information: total CPU time, user CPU time, system CPU time and real time. Similarly, the CPU rate graph shows

total CPU time as a percentage of real time, user CPU time as a percentage of real time, and system CPU time as a percentage of real time. The values (counts) are obtained by subtracting the respective cumulative values from the data returned by the "Return Resource Usage" remote procedure call at the start and end of the sample. The rates are computed by dividing the counts by the real time duration of the sample which is computed by subtracting the time stamps of the two "Return Resource Usage" calls.

The I/O usage graph shows four rows of information: the I/O blocks read, the I/O blocks written, messages read and messages written. The I/O rate graph shows the number of I/O blocks read per second, the number of I/O blocks written per second, the number of messages read per second and the number of messages written per second. The values (counts) are obtained by subtracting the respective cumulative values from the data returned by the "Return Resource Usage" remote procedure call issued at the start and end of the sample. Similarly, the rates are computed by dividing the counts by the real time duration of the sample, which is computed by subtracting the time stamps associated with the two "Return Resource Usage" calls.

The miscellaneous graph displays three elements: signals received, voluntary context switches and involuntary context switches. The miscellaneous rate graph shows the number of signals received per second, the number of voluntary context switches per second and the number of involuntary context switches per second. The values (counts) are obtained by subtracting the respective cumulative values from the data returned by the "Return Resource Usage" remote procedure call at the start and end of the sample. The rates are computed by dividing the counts by the real time duration of the sample which is computed by subtracting the time stamps of the two "Return Resource Usage" calls.

The displays will be updated in response to a specific user action or as new samples arrive. When a new sample arrives, if there is room on the time axis, it is added to the display. If the new sample would be off the screen and auto-scrolling is turned on, the time axis is scrolled to the left and the new sample added. The page display and resource usage windows naturally scroll together with the time axis. If auto-scrolling is disabled, the display remains unchanged, although the new data has been added to the pre-existing sample data. If the new sample has a logical sample of address space which is larger than previous samples or if it has a segment that was not present in earlier samples, the page display and page address window will be redrawn to fit the new segments in the display. If the resource usage data from a new sample exceeds the range of vertical scale for one of the resource usage graphs, a new scale will be set and the graph redrawn. When the user desires to end the monitoring, the quit button on the control and status panel or initialization panel is selected. Upon selection of the quit button, the monitoring process issues the "Stop Virtual Page Monitoring" remote procedure call to stop the virtual page monitoring process and release the operating system memory used to create the back copy of the page table. The data file is then closed and the connection to the operating system of the computer on which the monitored process is executing is terminated.

While the invention has been described in conjunction with the preferred embodiment, it is evident that numerous alternatives, modifications, variations and uses will be apparent to those skilled in the art in light of the foregoing description. In particular the method and apparatus of the present invention in not limited to use in the UNIX operating system environment and is not limited to a particular network. Furthermore, it is apparent to one skilled in the art that method and apparatus presented can be expanded to individually monitor multiple processes concurrently and display such information in a format easily understood by the user.

CLAIMS

1. In a computer system comprising resources such as a central processing unit (CPU), memory and input/output means, said computer system further comprising an operating system which manages the resources and interfaces processes executing on the computer with the resources, said operating system comprising tables and registers containing information indicative of the status of resources which are used to manage the resources, a method for extracting process performance information on a per-process basis, said method comprising the steps of:

providing on the computer system a network interface means;

supplying an operating system module comprising procedures that service remote procedure calls received through the network interface means, said procedures, when executed, retrieving specified information contained in the tables and registers that are used to manage the resources of the computer system;

during the execution of the process to be monitored, executing remote procedure calls which invoke the corresponding procedures in the operating system module which service the calls;

whereby the process performance information of the monitored process is extracted.

2. The method of claim 1;

wherein said procedures that service the remote procedure calls comprise:

a procedure that extracts specified information contained in the tables and registers at the beginning of a sample period; and

a procedure that extracts specified information contained in the tables and registers at the end of the sample period;

said method further comprising the step of computing the difference in the information taken at the beginning of the sample period and the end of the sample period to derive the performance information of the monitored process during the sample period.

3. The method of claim 1 wherein said network interface

formats according to the network protocol and transmits messages to a specified device through a network; and

receives formatted messages and extracts the message for use by the computer.

4. The method of claim 1 wherein

the monitored process is located on a first computer connected to a network through the network interface means;

the remote procedure calls are executed by a monitoring process located on a second computer which transmits the remote procedure calls over the network to the network interface means on the first computer;

said method further comprising

transmitting the retrieved information through the network interface means on the first computer through the network to the second computer;

receiving the retrieved information and storing the information on the second computer for subsequent access by a user;

whereby the monitoring process is executing on a computer system separate from the computer system on which the monitored process is executing, thereby minimizing the amount of resource usage on the first computer system required to extract process performance information of the monitored process.

5. The method of claim 4;

wherein said procedures that service the remote procedure calls comprise:

a procedure that extracts specified information contained in the tables and registers at the beginning of a sample period; and

a procedure that extracts specified information contained in the tables and registers at the end of the sample period;

said method further comprising the step of said monitoring process computing the difference in the information taken at the beginning of the sample period and the end of the sample period to derive the performance information of the monitored process during the sample period.

6. The method of claim 4 further comprising:

said monitoring process converting the retrieved data to a visual format; and

displaying the process performance information to the user.

7. The method of claim 1 wherein the remote procedure calls are executed by a separate monitoring process located on the computer system which transmits the remote procedure calls through the network interface means to the operating system module which services the remote procedure calls;

said method further comprising:

transmitting the retrieved information through the network interface means to the monitoring process;

storing the information for subsequent access by a user;

whereby the monitoring process is separate from the monitored process thereby eliminating the resource usage by the monitoring process that would be required to extract process performance information of the monitored process.

8. The method of claim 7:

wherein said procedures that service the remote procedure calls comprise:

a procedure that extracts specified information contained in the tables and registers at the beginning of a sample period; and

a procedure that extracts specified information contained in the tables and registers at the end of the sample period;

said method further comprising the step of said monitoring process computing the difference in the information taken at the beginning of the sample period and the end of the sample period to derive the performance information of the monitored process during the sample period.

9. The method of claim 7 further comprising:

said monitoring process converting the retrieved data to a visual format; and

displaying the process performance information to the user.

10. The method of claim 1;

wherein the remote procedure calls are executed by the monitored process located on the computer system which transmits the remote procedure calls through the network interface means to the operating system module which services the remote procedure calls;

said method further comprising:

transmitting the retrieved information through the network interface means to the monitored process;

storing the information for subsequent access by a user;

whereby the extraction of process performance information can be timed to precisely coincide with the execution of predetermined portions of the monitored process.

11. The method of claim 10;

wherein said procedures that service the remote procedure calls comprise:

a procedure that extracts specified information contained in the tables and registers at the beginning of a sample period; and

a procedure that extracts specified information contained in the tables and registers at the end of the sample period;

said method further comprising the step of said monitoring process computing the difference in the information taken at the beginning of the sample period and the end of the sample period to derive the performance information of the monitored process during the sample period.

12. In a virtual memory computer system comprising resources such as a central processing unit (CPU), memory and input/output means, said computer system further comprising an operating system which manages the resources and interfaces processes executing on the computer with the resources, said operating system comprising tables and registers containing information indicative of the status of resources which are used to manage the resources, said tables including page tables comprising read and write/modify bits for each process executing on the computer system, a method for extracting process performance information on a per-process basis, said method comprising the steps of:

providing on the computer system a network interface means;

supplying an operating system module comprising procedures that service remote procedure calls received through the network interface means, said procedures, when executed, retrieving specified information contained in the tables and registers that are used to manage the resources of the computer system, said procedures comprising:

a procedure that creates a back-copy of the page tables for the monitoring process;

a procedure that initiates monitoring of the page tables wherein the back-copy of the page tables is updated when the page tables are updated;

a procedure that clears the back-copy of the page tables independently of the page tables;

a procedure that returns a copy of the back-copy of the page tables to the monitoring process to be placed into a file;

during the execution of the process to be monitored, executing remote procedure calls which invoke the corresponding procedures in the operating system module which service the calls, said remote procedure calls comprising;

a remote procedure call to invoke the procedure to initiate monitoring of the page tables when it is desirable to begin a sample;

a remote procedure call to invoke the procedure which returns a copy of the back-copy of the page tables at the completion of the sample to the monitoring process to be placed into a file;

a remote procedure call to invoke the procedure which clears the back-copy of the page tables in order to prepare for a subsequent sample;

whereby the process performance information of the monitored process and samples of paging activity may be extracted.

13. The method of claim 12 wherein said network interface formats according to the network protocol and transmits messages to a specified device through a network; and

receives formatted messages and extracts the message for use by the computer.

14. The method of claim 12 wherein;

the monitored process is located on a first computer connected to a network through the network interface means;

the remote procedure calls are executed by a monitoring process located on a second computer which transmits the remote procedure calls over the network to the network interface means on the first computer;

said method further comprising

transmitting the retrieved information through the network interface means on the first computer through the network to the second computer;

receiving the retrieved information and storing the information on the second computer for subsequent access by a user;

whereby the monitoring process is executing on a computer system separate from the computer system on which the monitored process is executing, thereby minimizing the amount of resource usage on the first computer system required to extract process performance information of the monitored process.

15. The method of claim 14 further comprising:

said monitoring process converting the retrieved data to a visual format; and

displaying the process performance information to the user.

16. The method of claim 12;

wherein the remote procedure calls are executed by a separate monitoring process located on the computer system which transmits the remote procedure calls through the network interface means to the operating system module which services the remote procedure calls;

said method further comprising

transmitting the retrieved information through the network interface means to the monitoring process;

storing the information for subsequent access by a user;

whereby the monitoring process is separate from the monitored process thereby eliminating the resource usage by the monitoring process that would be required to extract process performance information of the monitored process.

17. The method of claim 16 further comprising:

said monitoring process converting the retrieved data to a visual format; and

displaying the process performance information to the user.

18. The method of claim 12;

wherein the remote procedure calls are executed by the monitored process located on the computer system which transmits the remote procedure calls through the network interface means to the operating system module which services the remote procedure calls;

said method further comprising

transmitting the retrieved information through the network interface means to the monitored process;

storing the information for subsequent access by a user;

whereby the extraction of process performance information can be timed to precisely coincide with the execution of predetermined portions of the monitored process.

19. In a virtual memory computer system comprising resources such as a central processing unit (CPU), memory and input/output means, said computer system further comprising an operating system which manages the resources and interfaces processes executing on the computer with the resources, said operating system comprising tables and registers containing information indicative of the status of resources which are used to manage the resources, said tables including page tables comprising read and write/modify bits for each process executing on the computer system, a method for extracting process performance information on a per-process basis, said method comprising the steps of:

supplying an operating system module comprising procedures that are invoked by and service system calls received, said procedures, when executed, retrieving specified information contained in the tables and registers that are used to manage the resources of the computer system, said procedures comprising:

a procedure that creates a back-copy of the page tables for the monitoring process;

a procedure that initiates monitoring of the page tables wherein the back-copy of the page tables is updated when the page tables are updated;

a procedure that clears the back-copy of the page tables independently of the page tables;

a procedure that returns a copy of the back-copy of the page tables to the monitoring process to be placed into a file;

during the execution of the process to be monitored, executing system calls which invoke the corresponding procedures in the operating system module which service the calls, said system calls comprising;

a system call to invoke the procedure to initiate monitoring of the page tables when it is desirable to begin a sample;

a system call to invoke the procedure which returns a copy of the back-copy of the page tables at the completion of the sample to the monitoring process to be placed into a file;

a system call to invoke the procedure which clears the back-copy of the page tables in order to prepare for a subsequent sample;

whereby the process performance information of the monitored process and samples of paging activity may be extracted.

20. A computer system comprising resources such as a central processing unit (CPU), memory and input/output means, said computer system further comprising an operating system which manages the resources, said operating system comprising tables and registers containing information indicative of the status of the resources which are used to manage the resources, a method for extracting process performance information on a per-process basis, said apparatus comprising:

a network interface means;

an operating system module comprising procedures that service remote procedure calls received through the network interface means, said procedures, when executed, retrieving specific information contained in the tables and registers that are used to manage the resources of the computer system;

during the execution of the process to be monitored, means for executing remote procedure calls which invoke the corresponding procedures in the operating system module which service the calls;

whereby the process performance information of the monitored process is extracted.

21. The apparatus of claim 20:

wherein said means that services the remote procedure calls comprise:

a procedure that extracts specified information contained in the tables and registers at the beginning of a sample period; and

a procedure that extracts specified information contained in the tables and registers at the end of the sample period;

means for computing the difference in the information taken at the beginning of the sample period and the end of the sample period to derive the performance information of the monitored process during the sample period.

22. A virtual memory computer system comprising resources such as a central processing unit (CPU), memory and input/output means, said computer system further comprising an operating system which manages the resources and interfaces processes executing on the computer with the resources, said operating system comprising tables and registers containing information indicative of the status of resources which are used to manage the resources, said tables including page tables comprising read and write/modify bits for each process executing on the computer system, means for extracting process performance information on a per-process basis comprising:

a network interface means;

an operating system module comprising procedures that service remote procedure calls received through the network interface means, said procedures, when executed, retrieving specified information contained in the tables and registers that are used to manage the resources of the computer system, said procedures comprising:

a procedure that creates a back-copy of the page tables for the monitoring process;

a procedure that initiates monitoring of the page tables wherein the back-copy of the page tables is updated when the page tables are updated;

a procedure that clears the back-copy of the page tables independently of the page tables;

a procedure that returns a copy of the back-copy of the page tables to the monitoring process to be placed into a file;

during the execution of the process to be monitored, means for executing remote procedure calls which invoke the corresponding procedures in the operating system module which service the calls, said remote procedure calls comprising;

a remote procedure call to invoke the procedure to initiate monitoring of the page tables when it is desirable to begin a sample;

a remote procedure call to invoke the procedure which returns a copy of the back-copy of the page tables at the completion of the sample to the monitoring process to be placed into a file;

a remote procedure call to invoke the procedure which clears the back-copy of the page tables in order to prepare for a subsequent sample;

whereby the process performance information of the monitored process and samples of paging activity may be extracted.

23. The system of claim 22 wherein said network interface formats information according to the network protocol and transmits messages to a specified device through a network, and receives formatted messages and extracts the information from the message for use by the computer.

24. The system of claim 22 wherein;

the monitored process is located on a first computer connected to a network through the network interface means;

the remote procedure calls are executed by a monitoring process located on a second computer which transmits the remote procedure calls over the network to the network interface means on the first computer;

said system further comprising:

means for transmitting the retrieved information through the network interface means on the first computer through the network to the second computer;

means for receiving the retrieved information and storing the information on the second computer for subsequent access by a user;

whereby the monitoring process is executing on a computer system separate from the computer system on which the monitored process is executing, thereby minimizing the amount of resource usage on the first computer system required to extract process performance information of the monitored process.

25. The system of claim 24 further comprising:

means for converting the retrieved data to a visual format; and

means for displaying the process performance information to the user.

26. A virtual memory computer system comprising resources such as a central processing unit (CPU), memory and input/output means, said computer system further comprising an operating system which manages the resources and interfaces processes executing on the computer with the resources, said operating system comprising tables and registers containing information indicative of the status of resources which are used to manage the resources, said tables including page tables comprising read and write/modify bits for each process executing on the computer system, means for extracting process performance information on a per-process basis comprising:

an operating system module comprising procedures that ar invoked by and service system calls, said procedures, when executed, retrieving specified information contained in the tables and registers that are used to manage the resources of the computer system, said procedures comprising:

a procedure that creates a back-copy of the page tables for the monitoring process;

a procedure that initiates monitoring of the page tables wherein the back-copy of the page tables is updated when the page tables are updated;

a procedure that clears the back-copy of the page tables independently of the page tables;

a procedure that returns a copy of the back-copy of the page tables to the monitoring process to be placed into a file;

during the execution of the process to be monitored, means for executing system calls which invoke the corresponding procedures in the operating system module which service the calls, said remote procedure calls comprising;

a system call to invoke the procedure to initiate monitoring of the page tables when it is desirable to begin a sample;

a system call to invoke the procedure which returns a copy of the back-copy of the page tables at the completion of the sample to the monitoring process to be placed into a file;

a system call to invoke the procedure which clears the back-copy of the page tables in order to prepare for a subsequent sample;

whereby the process performance information of the monitored process and samples of paging activity may be extracted.

27. A method for extracting process performance information on a per-process basis in a computer system substantially as hereinbefore described.

28. A computer system substantially as hereinbefore described with reference to the accompanying drawings.

Published 1991 at The Patent Office, State House, 66/71 High Holborn, London WC1R 4TP. Further copies may be obtained from Sales Branch, Unit 6, Nine Mile Point, Cwmfelinfach, Cross Keys, Newport, NP1 7HZ. Printed by Multiplex techniques ltd, St Mary Cray, Kent